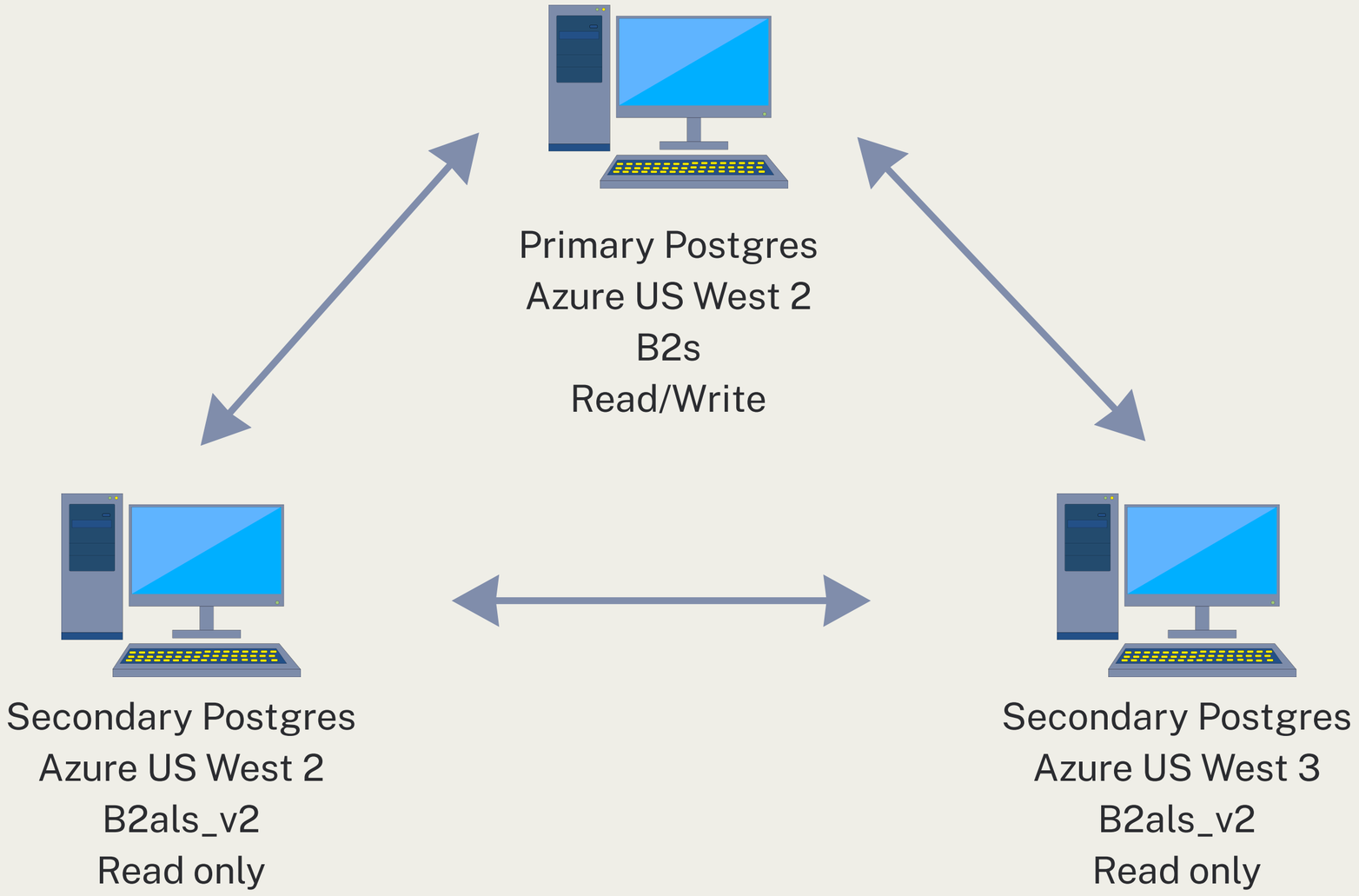
TimeScaleDB Cluster

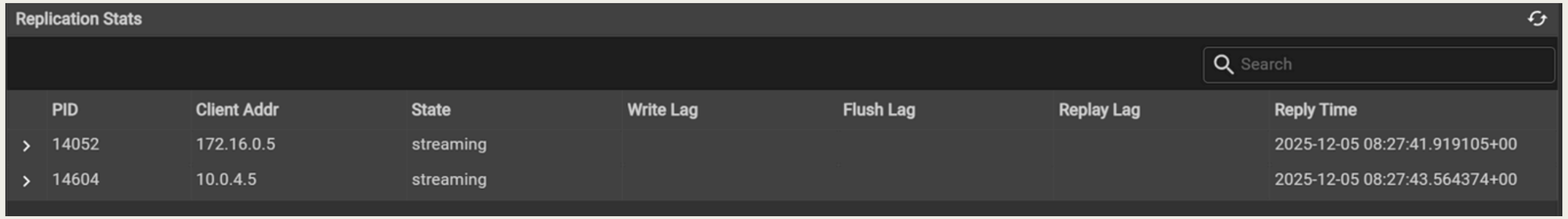
IS 497

Mohi & Ansh

INFRASTUCTURE OVERVIEW



REPLICATION




```
  pid   | usesysid |  usename   | application_name | client_addr | client_hostname | client_port |         backend_s
tart         | backend_xmin |   state    |  sent_lsn  | write_lsn  | flush_lsn  | replay_lsn | write_lag | flush_lag
 | replay_lag | sync_priority | sync_state |          reply_time
-------+----------+------------+------------------+-------------+-----------------+-------------+-----------------
-------------+--------------+-----------+------------+------------+------------+------------+-----------+----------
-+------------+---------------+------------+-------------------------------
 35444 |    16388 | replicator | 17/main          | 172.16.0.5  |                 |       49520 | 2025-12-05 15:00:
13.090626+00 |              | streaming | 0/BA703548 | 0/BA703548 | 0/BA703548 | 0/BA703548 |           |
 |            |             0 | async      | 2025-12-05 15:04:01.12819+00
 35336 |    16388 | replicator | 17/main          | 10.0.4.5    |                 |       47968 | 2025-12-05 14:57:
41.649248+00 |              | streaming | 0/BA703548 | 0/BA703548 | 0/BA703548 | 0/BA703548 |           |
 |            |             0 | async      | 2025-12-05 15:04:01.19626+00
(2 rows)
```

AUTOMATED BACKUP

```bash
#!/bin/bash

# Configuration
BACKUP_DIR="/var/backups/postgres"
DB_NAME="bitcoin_control"
DB_USER="postgres"
DATE=$(date +"%Y%m%d_%H%M%S")
FILENAME="$BACKUP_DIR/$DB_NAME-$DATE.sql.gz"

# 1. Perform the backup
# We use pg_dump and pipe it to gzip for compression
pg_dump -U $DB_USER $DB_NAME | gzip > $FILENAME

# 2. Check if backup was successful
if [ $? -eq 0 ]; then
  echo "Backup successful: $FILENAME"
else
  echo "Backup failed!"
  exit 1
fi

# 3. (Optional) Cleanup old backups
# This deletes files in the backup directory older than 7 days
find $BACKUP_DIR -type f -name "*.sql.gz" -mtime +7 -delete
~
```

Password managed using **`.pgpass`** file

```
#
# m h  dom mon dow   command
30 02 * * * /var/lib/postgresql/pg_backup.sh >> /var/log/pg_backup.log 2>&1
~
```

Cron job setup for automated backup

```
azureuser@Mohi-Postgres1:~$ ls -ltr /var/backups/postgres/
total 46776
-rw-rw-r-- 1 postgres postgres 23948162 Dec  5 12:26 bitcoin_control-20251205_122557.sql.gz
-rw-rw-r-- 1 postgres postgres 23948160 Dec  5 12:26 bitcoin_control-20251205_122623.sql.gz
```

SECURITY

User Account	Scope & Privileges	Security Controls
postgres **(Superuser)**	Initial setup & maintenance only. Full system access.	Application access prohibited. Protected by strong MFA.
replicator **(System)**	Data replication only. REPLICATION privilege allowed.	No SQL read/write access. Restricted to Replica IPs.
application **(Service)**	Backend read/write operations. SELECT, INSERT on schema.	No DDL (DROP/ALTER) allowed. Security Group Whitelisting.
btc_service **(Integration)**	Writes to bitcoin_db & bitcoin_control. INSERT permissions.	No DDL (DROP/ALTER) allowed.

DATASET OVERVIEW

Name: 2024 High-Frequency Bitcoin Market Data

Details:

- Source: Binance Public Data Collection (Spot Market).
- Asset Pair: Bitcoin vs. Tether (BTC/USDT).
- Granularity: 1-Minute Intervals (OHLCV).
- Time Range: January 1, 2024 – December 31, 2024 (1 Full Year).
- Data Volume: ~525,600 rows of high-precision financial data.
- Schema:
  - Timestamp: Unix Epoch (Milliseconds).
  - Metrics: Open, High, Low, Close Price, Trading Volume.

DATA EXTRACTION STRATEGY

Automated Data Pipeline

- Challenge: Data was fragmented into 12 separate monthly ZIP files on a remote server.
- Solution: Developed a Python automation script.
- Process:
    a. Automated Retrieval: Script iterates through months (01–12) for the year 2024.
    b. Dynamic Extraction: Fetches ZIP files from Binance and extracts CSVs on-the-fly.
    c. Storage: Organized raw files into a dedicated directory (btc_2024_data) for processing.

DATA LOADING & INGESTION

Optimizing Ingestion for Cloud Infrastructure

- Infrastructure: Azure B2s VM (2 vCPUs, 4GB RAM).

- Challenge:
  - Bulk loading the entire year triggered "Out of Shared Memory" errors due to PostgreSQL lock limits.
  - We faced this issue first on a B1 server and got the same error even after upgrading to a B2als_v2 server with increased RAM and disks.
  - Tried importing the 2024 and 2025 datasets using batch imports but the machine could not handle the entire data.

- Final Approach: Loaded only 2024 data in batches to ensure the server did not crash.

COMPARATIVE STUDY - AGGREGATION PERFORMANCE

Test 1: Arbitrary Time Bucketing (15-Min Intervals)

Objective: Group 525,000 rows into 15-minute buckets to calculate volume & volatility

Standard Postgres:

- Query: Requires complex math: floor(open_time / 900) * 900.
- Runtime: ~14 Seconds.

TimescaleDB:

- Query: Uses native optimization: time_bucket('15 minutes', time).
- Runtime: ~6 Seconds.

Result: TimescaleDB was 2.3x Faster for time-based grouping.

COMPARATIVE STUDY - ANALYTICAL FUNCTIONS

Test 2: Daily Net Change (First vs. Last)

Objective: Calculate the daily price movement (Close - Open) for every day in 2024

Standard Postgres:

- Limitation: Lacks first() and last() functions.
- Workaround: Must sort all rows in memory using expensive arrays.
- Runtime: 0.9 Seconds

TimescaleDB:

- Optimization: Uses native first() and last() functions.
- Efficiency: Instantly grabs boundary values without sorting the entire dataset.
- Runtime: 0.37 Seconds

Result: Significant reduction in query complexity and execution resources.

COMPARATIVE STUDY - SYNTAX & EFFICIENCY

Test 3 - The "First & Last" Problem (Weekly OHLC)

Objective: Calculate the weekly Open, High, Low, and Close (OHLC) prices for the entire year of 2024.

Standard Postgres:

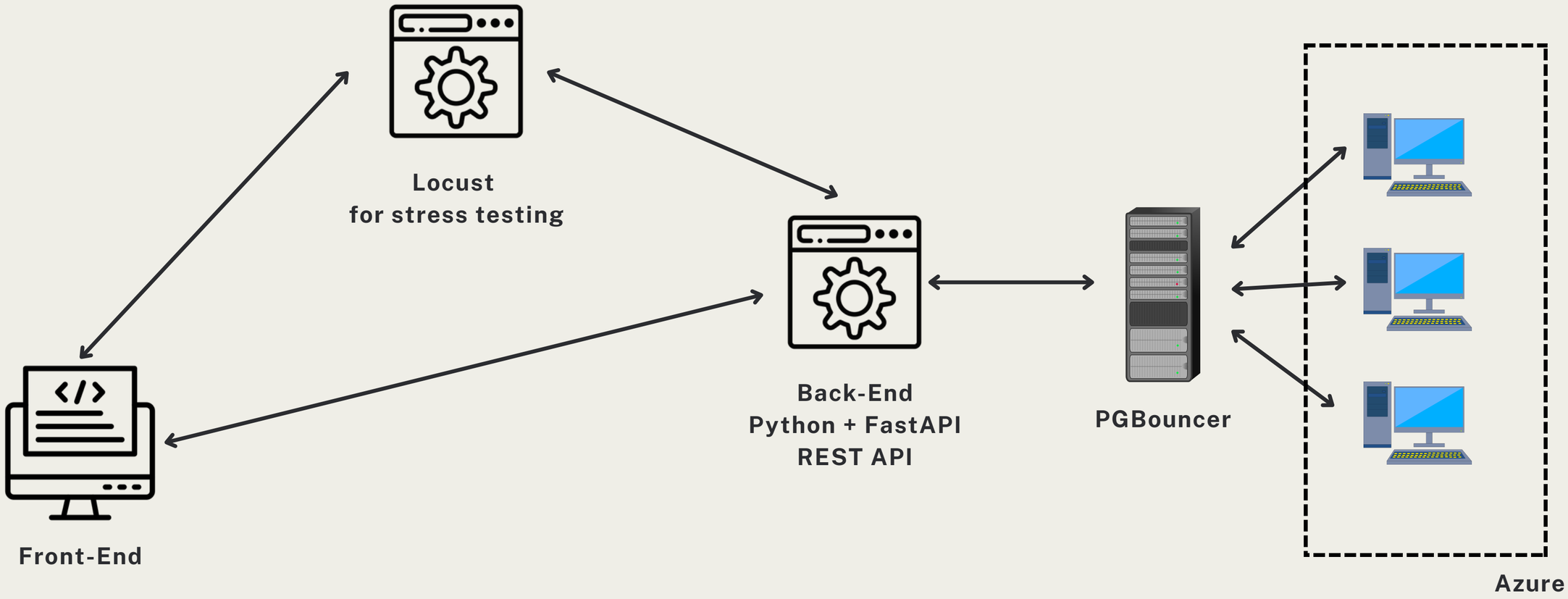
- Limitation: Lacks first() and last() functions.
- Workaround: Must build an array of all prices in a week, sort them in memory, and pick the first one.
- Runtime: 1.83 Seconds

TimescaleDB:

- Optimization: Uses native first() and last() functions.
- Efficiency: Scans the index to grab the boundary values instantly without sorting intermediate data.
- Runtime: 0.31 Seconds

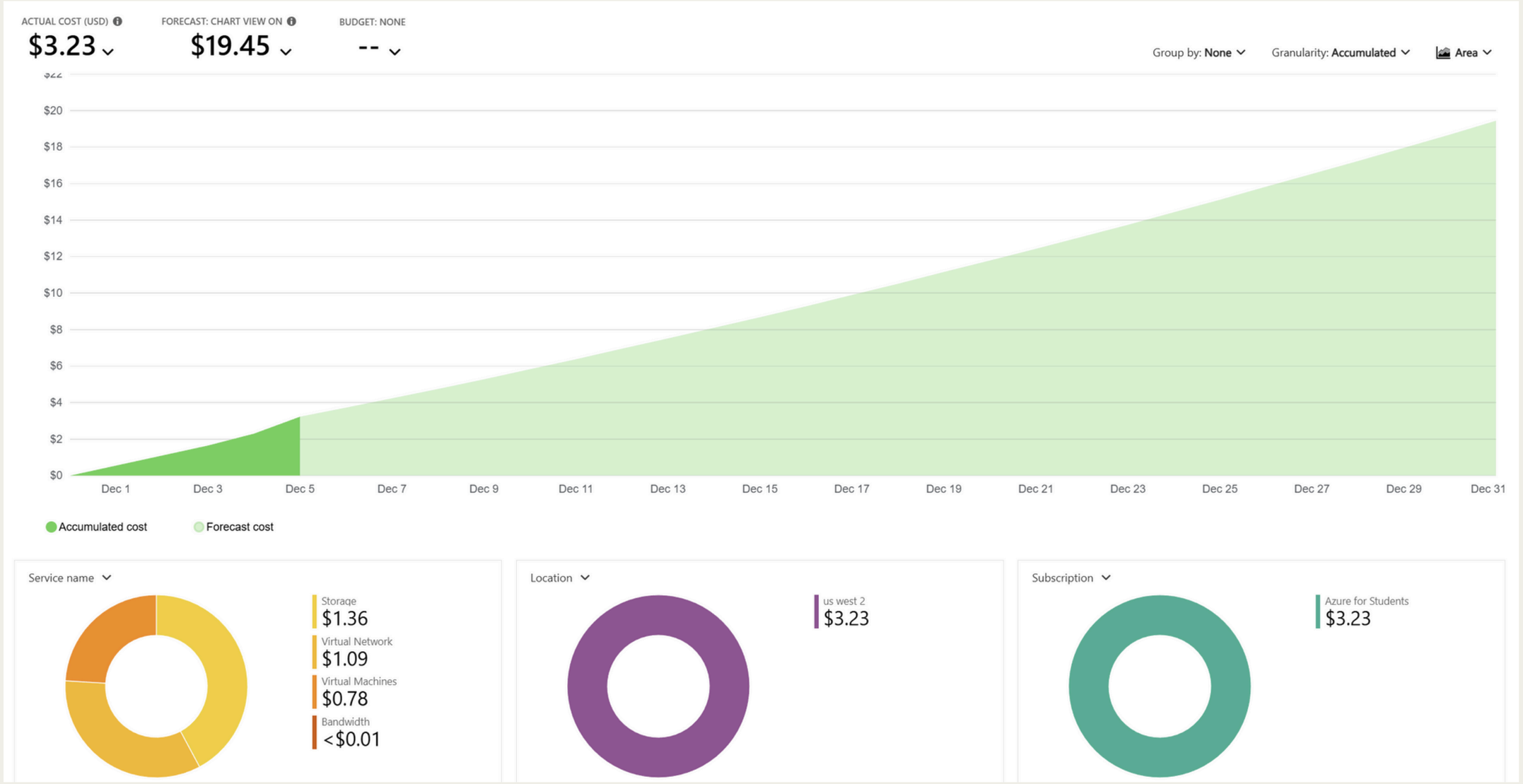
Result: TimescaleDB was Significantly Faster and required far less memory (RAM)

FULL STACK APPLICATION



D E M O

COSTING


ACTUAL COST (USD)
$3.23
FORECAST: CHART VIEW ON
$19.45
BUDGET: NONE
--
Group by: None
Granularity: Accumulated
Area
$20
$18
$16
$14
$12
$10
$8
$6
$4
$2
$0
Dec 1
Dec 3
Dec 5
Dec 7
Dec 9
Dec 11
Dec 13
Dec 15
Dec 17
Dec 19
Dec 21
Dec 23
Dec 25
Dec 27
Dec 29
Dec 31
Accumulated cost
Forecast cost
Service name
Storage
$1.36
Virtual Network
$1.09
Virtual Machines
$0.78
Bandwidth
<$0.01
Location
us west 2
$3.23
Subscription
Azure for Students
$3.23

COSTING

Resource Component	Specification	Est. Monthly Cost
Database Compute (3 Nodes)	3x Standard_B2ms (2 vCPU, 8GB)	$200.00
Storage (Managed Disks)	3x 512GB Premium SSD (P20)	$200.00
Backup Storage	Azure Blob Storage	$15.00
Data Transfer	Inter-VNet & Outbound Traffic	$25.00
Total Estimated Cost	**Azure Production Setup**	**~$440.00 / mo ($5280 / year)**

Thank you!